

SAP AG

21. Okt. 2004

Intellectual Property
Department

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property
Organization
International Bureau



(43) International Publication Date
14 October 2004 (14.10.2004)

PCT

(10) International Publication Number
WO 2004/088516 A1

(51) International Patent Classification⁷: G06F 9/45, 9/44

(21) International Application Number:
PCT/EP2004/001382

(22) International Filing Date: 13 February 2004 (13.02.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
03007527.9 1 April 2003 (01.04.2003) EP

(71) Applicant (for all designated States except US): SAP
AKTIENGESELLSCHAFT [DE/DE]; Neurottstrasse
16, 69190 Walldorf (DE).

(72) Inventors; and

(75) Inventors/Applicants (for US only): BARTSCH, Eduard
[DE/DE]; Kolpingstr. 14, 68766 Hockenheim (DE). LIEN-
HARDT, Joerg [DE/DE]; Haydenstrasse 17, 69190 Walldorf (DE).

(81) Designated States (unless otherwise indicated, for every
kind of national protection available): AE, AG, AL, AM,

AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN,
CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI,
GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE,
KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD,
MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG,
PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM,
TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM,
ZW.

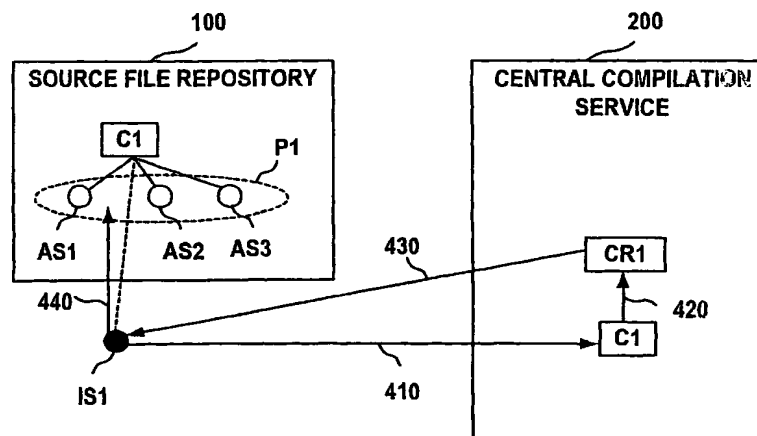
(84) Designated States (unless otherwise indicated, for every
kind of regional protection available): ARIPO (BW, GH,
GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW),
Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), Euro-
pean (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR,
GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK,
TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,
ML, MR, NE, SN, TD, TG).

Published:

— with international search report

For two-letter codes and other abbreviations, refer to the "Guid-
ance Notes on Codes and Abbreviations" appearing at the begin-
ning of each regular issue of the PCT Gazette.

(54) Title: METHOD AND COMPUTER SYSTEM FOR ACTIVATION OF SOURCE FILES



900

(57) Abstract: Method and computer system (900) for activation of source files. A source file repository (100) stores a plurality of active source files (AS1, AS2, AS3) belonging to a component (C1). A central compilation service (200) receives (410) an activation request for at least one inactive source file (IS1) of the component (C1). In response to the activation request the central compilation service (200) compiles (420) the component (C1) using the at least one inactive source file (IS1) and, in case the compilation is successfully completed, initiates (430) a transfer (440) of the at least one inactive source file (IS1) to the plurality (P1) of active source files.

WO 2004/088516 A1

WO 2004/088516

PCT/EP2004/001382

METHOD AND COMPUTER SYSTEM FOR ACTIVATION OF SOURCE FILESField of the Invention

- 5 The present invention generally relates to electronic data processing, and more particularly, relates to methods, computer program products and systems for file based software development.

Background of the Invention

- 10 Many software products are built from a large number of source files that are usually written in a programming language that requires the compilation of the sources into a format that can be interpreted by computers efficiently. Often the software products are developed
15 by large groups of software developers.

- Software developers are usually equipped with computers that run integrated development environments (IDEs), such as Eclipse or Forte, to provide functionality for editing/modifying and compiling
20 source files residing on the local hard drive of the developer's computer.

- In order to allow cooperation in teams, source files are stored centrally and versioned using a source control mechanism (e.g., CVS) that contains the source
25 files. A developer downloads source files from a source control system to his local computer, changes the source files and then stores the changed source files again in the source control system. A further developer can download the changed source files to his/her local
30 computer by downloading them from the source control system.

To test changed source files, a developer can compile them on the local computer. To test interoperability with further source files that are

changed by other developers, the developer has to compile the further changed source files as well.

Computing power of current computer hardware is insufficient to allow instant recompilation of large software products on a developer's local computer within an acceptable amount of time. Therefore, some IDEs, such as Eclipse, provide the ability to test changed source files of a software product by splitting the source files into smaller groups. These smaller groups can be compiled without a requirement to compile the software product as a whole. The smaller groups are typically referred to as projects or components. In the following description the term component is used.

Compilation and provision of distributable compilation results of the whole software product is typically performed by a central computer (e.g., a server computer). Typically, the central compilation is done after having transferred the source files of all components from the source control system to the central computer once a day, every few hours or in similar large time intervals. The central computer compiles all components and stores the compilation results. However, compiling all components even if no source files have been changed is a waste of resources of the central computer that might be usable otherwise.

Further, a developer always has to wait for the next central compilation before he/she can test the interoperability of the compilation result of a changed source file.

Further, the developer can transfer changed source files to the central computer that may cause errors in the central compilation although a previous local test compilation worked properly because of using outdated local copies of the compilation results of referenced components. If the central compilation fails each developer has to wait one more compilation interval before updating local copies of used compilation

results (e.g., used libraries). If further changes of source files occur in the meantime, there is an increased probability that the central compilation fails due to these changes, because in many cases the
5 changes have been tested against outdated libraries on local computers. This probability increases with the number of developers that work on the software product.

There is an ongoing need to reduce the number of central compilation failures.

10

Summary of the Invention

Therefore, the present invention provides methods, computer program products and computer systems as described by the independent claims to anticipate
15 whether changed source files will make a subsequent central compilation fail or not.

To achieve this objective a central compilation service checks any modification of inactive source files that belong to a component to identify whether
20 the modifications are successfully compilable or not. If the compilation can be successfully completed modified inactive source files are activated and become active source files of the component.

Active source files become visible in an
25 integrated development environment (IDE) for each software developer. It is an effect of the present invention that source files of a component can only be activated if the component is compilable.

A further effect of the present invention is to
30 provide a general (generic) component- and dependency-format and to generate IDE-specific component definitions based on the general formats. This allows one to integrate and use any IDE in the development of large software products.

35 A further effect of the present invention is to provide an IDE independent compilation tool that allows

one to plug-in compilation mechanisms for different programming languages, runtime systems, etc., to be used in a central compilation. Thus, a compilation can be started from within an IDE but also outside the IDE.

- 5 A further effect of the present invention is to provide a parallel build mechanism by determining subsets of components that do not depend on each other and compile modified subsets in parallel.

Embodiments of the present invention help to:

- 10 a) reduce the waiting time of a developer for getting up-to-date results from the central compilation service because compilation on request by using an incremental build mechanism eliminates the developer's dependency on a certain cyclic compilation schedule;
- 15 b) quickly notify a software developer, who changed an inactive source file causing problems for a successful central compilation;
- c) reduce the number of compilation failures when using central compilation because the concept of active
- 20 source files allows one to activate only changes of source code that do not cause compilation errors when compiling the corresponding component;
- d) speed up the compilation of components by using a parallel build mechanism; and
- 25 e) reduce the production cost of software products as a consequence of a) to d).

The aspects of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims. Also,

30 the described combination of the features of the invention is not to be understood as a limitation, and all the features can be combined in other constellations without departing from the spirit of the invention. It is to be understood that both, the foregoing general

35 description and the following detailed description are exemplary and explanatory only and are not restrictive of the invention as described.

Brief Description of the Drawings

- FIG. 1 is a simplified block diagram of a computer
5 system that can be used with an embodiment of
the invention for activating source files;
FIG. 2 illustrates a further embodiment of the
computer system;
FIG. 3 illustrates notification of a user/owner by the
10 computer system in case of compilation failure;
FIG. 4 illustrates the use of a change list with one
embodiment of the invention;
FIG. 5 is a simplified block diagram of the computer
system to illustrate a software development
15 process using the spirit of the invention;

Detailed Description of the Invention

The same reference numbers are used throughout the
20 drawings to refer to the same or like parts.
Definitions of terms, as used herein after:

Active source file:
source file that was previously activated.

25 Inactive source file:
source file that has been edited/modified and is
subject to subsequent activation.

30 Compilation:
translating source files into runtime files. This may
also include transforming a formatted text file into a
corresponding HTML file. It may also include to

transform files from or to formats such as XML, forms,
etc.

Formatted

Activation:

- 5 transferring an inactive source file to a plurality of
active source files.

Run time archive:

- a pool for compilation results (runtime files) of
10 components.

Central compilation service:

- a compilation service, for example implemented as a
J2EE based server-side database application, compiles
15 source files into assemblies, such as JAR files or EAR
files. A JAR file has a file format based on the ZIP
file format and is used for aggregating many files into
one. An Enterprise Application Archive (EAR) file has a
format used for software distribution by a J2EE
20 platform.

FIG. 1 is a simplified block diagram of a computer
system 900 that can be used with an embodiment of the
invention for activating source files of a component.

- 25 The computer system 900 includes a source file
repository (SFR) 100 that stores a plurality P1 of
active source files AS1, AS2, AS3 belonging to a
component C1 (illustrated by solid connection lines
between the component and each active source file of
30 the component). For example, the SFR 100 can be
implemented as a file system.

- An active source file of a component is a source
file that has been used in a successful compilation of
the component C1 and, as a consequence, has been
35 activated. For example, active source files can be made

visible through an IDE to any user who might be interested in using any of the active source files. For example, the users are software developers, where each software developer works on a set of components.

5 Further, active source files of a component are consistent with each other with respect to the component. That is, when the component is compiled on the base of its active source files there are no compilation errors to be expected. However, the

10 totality of active source files in SFR 100 is not necessarily in a state that allows one the successful compilation of all contained sources. In case of component dependencies it may occur that the active source files of one component are not compatible with a

15 further component that uses component. This allows a first software developer to intentionally introduce changes to the component that lead to incompatibilities with the further component and let a second software developer react on these changes later on, thus

20 bringing a high degree of flexibility to the software development process.

The component C1 can also have inactive source files (e.g., inactive source file IS1), that may be stored elsewhere (illustrated by a dotted connection

25 line between the component C1 and the inactive source file IS1). For example, the inactive source files may be stored on a local computer of a software developer. An inactive source file of the component is a source file that has been changed since the last successful

30 compilation of the component and that has not been activated, yet. That is, an inactive source file has been created or modified after the last successful compilation of the component that includes the inactive source file. Optionally, inactive source files can be

made visible through an IDE to any user who might be interested in using any of the inactive source files.

The computer system 900 further includes a central compilation service 200. For example, CCS 200 can be
5 implemented as a server-side J2EE based database application. When the central compilation service (CCS) 200 receives 410 an activation request for at least one inactive source file IS1 of the component C1, CCS 200 compiles 420 the component C1 using the at least one
10 inactive source file IS1. For example, a user can trigger the activation request when having completed a modification leading to the inactive source file IS1. For example, the user can execute a corresponding function to activate the inactive source file IS1. The
15 activation request can also relate to further inactive source files that may be activated together with the inactive source file IS1.

In case the compilation is successfully completed, that is, the component C1 is successfully compiled 420
20 into the compilation result CR1, CCS 200 initiates 430 a transfer 440 of the at least one inactive source file IS1 to the plurality P1 of active source files. In other words, the at least one inactive source file IS1 becomes an active source file. If further inactive
25 source files were used in the error free compilation 420, they are also transferred to the plurality P1 of active source files, thus becoming active source files.

The transfer 440 of an inactive source file can depend on various change modes.

30 For example, in change mode "adding", a new inactive source file has been created. Therefore, a corresponding active source file does not exist, yet. As a consequence, once the new inactive source file gets activated, a corresponding new active source file
35 is added to the plurality P1 of active source files.

In change mode "replacing", an inactive source file (e.g., IS1) that corresponds to an active source file is modified. When the inactive source file is activated, the corresponding active source file is replaced by the activated source file.

In change mode "deleting", an inactive source file is deleted. As a consequence, the corresponding active source file is deleted from the plurality P1 of active source files.

10 If multiple inactive source files are activated simultaneously, the transfer of each of the inactive source files can depend on a different change mode.

At a point in time when all inactive source files of a component are successfully activated, a user can rely on an error free compilation of the changes previously applied to the corresponding inactive source files. An inactive source file has to be activated when modified, deleted from or added to the component. The inactive source file is not necessarily consistent with the active source files of the component. Activating the inactive source file after a successful test compilation of the component including the inactive source file gives the user certainty about the consistency of the activated (previously inactive) source file with the other active source files of the component. If the test compilation is not successfully completed, the source file remains inactive.

FIG. 2 illustrates a further embodiment of the computer system 900.

30 In the further embodiment the computer system 900 includes a runtime archive storage 300 to store 450 the compilation result CR1 of the component C1 in case the compilation of the component C1 is successfully completed. For example, when a larger portion of the

software product or the software product as a whole is subject to compilation, components that have a corresponding compilation result stored in the runtime archive storage (RTA) 300 do not have to be recompiled if no activation directly or indirectly affected the components since their last compilation.

Further, the computer system 900 includes a further source file repository 100' to store the at least one inactive source file IS1. In other words, the further embodiment introduces the further SFR 100' as a development depot of inactive source files, whereas SFR 100 can be considered as a pool of active source files. Preferably, each active source file of the pool has a corresponding inactive source file in the development depot. The development depot and the pool can be part of a version control system for source files. The transfer of code changes from the development depot to the pool is tied to successful compilation of the components that are affected by the changes to be activated.

For example, a software developer checks in changes into the development depot SFR 100', that is, the software developer has edited an inactive source file (e.g., IS1). To make the changes visible to other software developers the software developer triggers an activation request for the inactive source file IS1 of the component C1. CCS 200 compiles 420 the component C1 using the inactive source file IS1.

In case of component dependencies, CCS 200 first determines all components affected by the change and then performs a test compilation of the affected components using the corresponding active source files and the inactive source files to be activated. In the example, the component C0 depends D1 on the component C1 and the component C1 depends D2 on the component C2.

CCS 200 evaluates the component dependencies D1, D2. If, for example, the component C2 is also affected by the changes that are to be activated, the component C2 is also compiled 421 in response to the activation request. The compilation result CR2 of the component C2 can also be stored 451 in the RTA 300 in case of successful compilation. A further example, where the component C2 is not subject to compilation is explained in FIG. 4.

10 In case of successful compilation of all affected components the corresponding inactive source files are added to SFR 100, and the compilation results of the affected components are stored centrally in RTA 300.

The CCS 200 can assign a component status to each component depending on the result of the compilation. Examples of component status are:

"ready" in case the compilation of the component is successfully completed;

20 "broken" in case the compilation of the component fails; and

"dirty" in case the component depends on a further component and the compilation of the further component is successfully completed.

In the above component dependency example, CCS 200 assigns the component status "ready" to the components C1 and C2, whereas the component status "dirty" is assigned to the component C0 because it depends D1 on the component C1 that was successfully compiled. Components that have the component status "dirty" or 30 "broken" will be automatically considered in a later compilation run.

In another embodiment of the invention, CCS 200 can enforce consistency of the active source files across all components of the software product.

In this embodiment the compilation of all components that depend directly or indirectly on changed components (components that include changed inactive source files) is done together with the
5 activation of the inactive source files. In other words, an activation fails in this embodiment if any of the changed components can not be compiled or any component that directly or indirectly depends on the changed components can not be compiled. In this
10 embodiment, for example, incompatible interface changes are activated simultaneously with the correct adaptation of all source files that make use of these interfaces.

15 FIG. 3 illustrates notification of a user 90 who is the changer of the inactive source file IS1 (illustrated by dotted line) in case of compilation failure.

In this example, the inactive source file IS1 that is subject to an activation in response to an
20 activation request is not consistent with active source files or further inactive source files being used in the (test) compilation of the corresponding component(s) C1. For example, a function signature in the inactive source file does not comply with a call of
25 the function by an active source file within the same component, or a type of a variable in the inactive source file has been modified and does not comply anymore with the type declaration in an active source file within the same component. In this case the
30 subsequent compilation of the component fails 460 and no compilation result is created (illustrated by crossed out CR1). Therefore, CCS 200 keeps the RTA 300 consistent with SFR 100.

CCS 200 then notifies 470 the changer 90 of the
35 inactive source file IS1 that caused the compilation

failure. If multiple inactive source files create problems and the inactive source files are owned by different users, each user can be notified accordingly. For example, CCS 200 can send a corresponding e-mail
5 message to each user or the messages are listed in a central failure report where each user can retrieve the messages affecting his/her inactive source files.

The messages inform the users about the compilation errors and enable each user to correct the corresponding errors in the defective inactive source
10 files before sending further activation requests.

The (test) compilation in response to the activation request allows CCS 200 to anticipate whether an inactive source file will make a subsequent central
15 compilation fail or not. CCS 200 protects the pool of active source files from inconsistencies because an inactive source file that causes compilation errors is not activated and, therefore, not transferred to the plurality P1 of active source files.

20
FIG. 4 illustrates the use of a change list CL1 with one embodiment of the invention. The example of FIG. 4 is based on the embodiment described in FIG. 2, where inactive source files are stored in the further SFR
25 100'. Local storages on local computers can be used instead.

The change list CL1 stores representations of inactive source files that have been modified since a previous compilation of the component C1. Multiple
30 change lists may exist. For example, there can be a change list for every user who is modifying source files. In another implementation there can be change lists by components.

CCS 200 receives the activation request 601 that
35 targets all inactive source files having a

representation in the change list CL1. CCS 200 includes a component dependency evaluator (CDE) 210 that determines, which components need to be considered when compiling the component C1 in response to the activation request 601. In the example, the component C1 depends D2 on component C2.

CDE 210 ensures that all building elements that are required for a compilation of the component C1 are provided (straight dashed arrows) to CCS 200. For example, this can be achieved by retrieving 520 the plurality P1 of active source files of C1 from SFR 100. Further, the inactive source files having a representation in the change list CL1 are loaded 510 from the further SFR 100'. For example, the loaded inactive source files overwrite, delete or replace the corresponding active source files of the plurality P1. The result is a second plurality P2 of source files that are the basis for the subsequent compilation. Further, the compilation result CR2 of the component C2 is retrieved 530 from RTA 300. As a result, CCS 200 has all building elements available to compile the component C1 into the compilation result CR1.

The compilation of the component C1 depending D2 on further components (e.g., component C2) will be referred to as "incremental build" if previously obtained compilation results (e.g., CR2) of the further components (e.g., C2) can be reused in the compilation of the component C1. In this case, only the "increment", that is, the second plurality P2 needs to be compiled. In another implementation of "incremental build" compilation results can be provided at the level of source files. In this implementation, only the inactive source files of the component C1 and those active source files whose compilation results are directly affected by the inactive source files need to

be compiled. For non-affected source files the corresponding compilation result can be used. In this implementation the increments are smaller.

In another implementation of the invention, CCS
5 200 performs a "parallel build" when compiling the component C1. CDE 210 evaluates dependencies of the component C1 on further components. In case no dependencies exist, the component C1 and the further components can be compiled in parallel.

10 Because of hardware limitations, such as limited processor speed or limited data throughput of memory devices, a parallel build can be performed by a cluster of build computers. In this case, the compilation of various components is distributed on multiple computers
15 that are all controlled by CCS 200. This allows CCS 200 to speed up the parallel build by using the processor and memory resources of all build computers simultaneously.

20 FIG. 5 is a simplified block diagram of the computer system 900 to illustrate a software validation process using the spirit of the invention.

The separation of source files of one software product into a plurality of components usually leads to
25 a situation where source files from one component depend on contents of other components, because there are, for example, references defined between them. To allow a developer a local compilation of a single component on his/her computer 901, the compilation
30 results of the referenced components are usually used instead of the corresponding source files. This allows one to avoid a full recompilation of the whole software product on every developer's computer, which would occur if the source files of referenced projects were
35 used instead of the compilation results.

Usually, a developer locally compiles only source files of a component modified by the developer. Therefore, the developer obtains 1b compilation results of referenced components from somewhere else (e.g.,
5 from RTA 300 on a central computer 903) and stores these compilation results in his/her local file system 110. The central computer 903 can communicate with the local computer 901 over a network 999, such as a local area network (LAN), wide area network (WAN) or the
10 Internet.

Especially for larger software products it is advantageous to handle the creation and distribution of compilation results in a centralized manner. To efficiently handle these compilation results if stored
15 locally on further local computers is difficult. Further, the central storage of the compilation results enables developers to get the compilation results of all components from one location instead of from potentially all local development computers. Centrally
20 produced compilation results can also be used to install and test the whole software product.

A source control system can also be implemented on a central location, such as computer 902 storing source file repositories. The computer 902 can communicate
25 with further computers of the computer system 900 over the network 999. For example, the source control system can be implemented as a file system implementing the Web-based Distributed Authoring and Versioning (WebDAV) protocol or the DeltaV protocol. The source control
30 system can support conventional source control functions, e.g., change tracking, merging of document versions, change propagation mechanisms and automatic conflict detection, known by those skilled in the art. Change tracking means that when a version of a source
35 file is created it is stored in an activity. An

activity is the smallest granularity object for propagating changes between source file repositories. A new version that is contained in an open activity is only visible for the creator. After the activity gets
5 checked into the source control system, the version becomes visible for everybody.

When a developer wants to edit (EDIT) a source file, the source file can be retrieved 1a from a source file repository of the source control system. In one
10 implementation, general (generic) component- and dependency-descriptions are used for describing components and their dependencies, and these descriptions are also stored in the source control system. When retrieving source files and component
15 descriptions from the source file repositories, the development computer 901 converts general component description formats to IDE-specific component definitions and dependency definitions that are appropriate for an IDE 120 used by the computer 901.

20 Then, the source file is stored in the local file system 110 and can be edited by the developer using the IDE 120. Once a source file is modified it can be transferred 2 from the IDE 120 to a local build tool 140. The local build tool 140 further retrieves 3
25 necessary compilation results from the local file system 110. The compilation results may have been previously obtained 1b from RTA 300. Then, the local build tool 140 locally compiles 4 components that include the modified source file(s) by using the
30 necessary compilation results. If the local compilation is successful the new compilation result is stored 5 in the local file system 110 from where it can be deployed 6 by the IDE 120 to a local runtime 130 for test purposes.

Then, the developer can check in 7 the modified source file(s) into a source file repository in the computer 902. The modified source file(s) become inactive source file(s) that need(s) to be activated.

5 The developer can trigger an activation request with regards to the inactive source file(s). For example, the activation request may be launched 8 through the IDE 110 and is directed to the central computer 903 that includes CCS 200.

10 Once CCS 200 receives the activation request it loads 9a corresponding active or inactive source files from the source file repositories 902 and retrieves 9b corresponding compilation results from RTA 300.

Then, CCS 200 centrally compiles 10 the component(s) affected by the activation request. In case of successful central compilation 10 the compilation result(s) of the affected component(s) are made available 11 in RTA 300. Further, CCS 200 triggers 12 the activation of the successfully compiled inactive source file(s) in the source file repositories 902.

20 From then onwards, the modifications that were made during the editing (EDIT) activity of one developer are available on the central source file repositories and can be used by any other developer.

25 In case an error occurs during the (test) compilation 10, steps 11 and 12 are not executed but the error result is stored and made available to the developer(s) who are responsible for the corresponding modifications. For example, the IDE that has triggered 30 8 the activation request can check the status of the activation. If the activation failed the developer will recognize this and is now able to correct the problem immediately. Even if the developer does not recognize the problem (e.g., starts activation and leaves), the 35 pool of active source files is not corrupted, because

the modifications do not become effective there. As a result, all other developers working on the software product are not disturbed by failing compilations (caused by others), which increases the productivity of the software development process. In this synchronous implementation the waiting time of a developer is reduced primarily to the time of the compilation run.

In another asynchronous implementation, instead of a developer triggering the activation request, a program constantly checks the source control system for new checked-in modified (inactive) source files. In this case the activation request is generated by the program, for example, if new inactive source files exist. The central (test) compilation is started after having received the activation request. Additionally, the program determines from the inactive source files which components have to be compiled. This is achieved by providing the possibility to define dependencies between components. The developer's waiting time is one central compilation cycle that is determined by the intervals used by the program to check the source control system.

In both the synchronous and the asynchronous implementation repair cycles for defective components or source files is reduced from one day in a prior art scenario where a central build of all components is run overnight and errors become visible on the next day to less than an hour or even several minutes.

The invention can be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. The invention can be implemented as a computer program product, i.e., a computer program tangibly embodied in an information carrier, e.g., in a machine-readable

storage device or in a propagated signal, for execution by, or to control the operation of, data processing apparatus, e.g., a programmable processor, a computer, or multiple computers. A computer program can be
5 written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A
10 computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a communication network.

Method steps of the invention can be performed by
15 one or more programmable processors executing a computer program to perform functions of the invention by operating on input data and generating output. Method steps can also be performed by, and apparatus of the invention can be implemented as, special purpose
20 logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application-specific integrated circuit).

Processors suitable for the execution of a computer program include, by way of example, both
25 general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. The essential elements of a computer are at
30 least one processor for executing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for
35 storing data, e.g., magnetic, magneto-optical disks, or

optical disks. Information carriers suitable for embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, e.g.,
5 EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto-optical disks; and CD-ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in special purpose logic circuitry.

10 To provide for interaction with a user, the invention can be implemented on a computer having a display device, e.g., a CRT (cathode ray tube) or LCD (liquid crystal display) monitor, for displaying information to the user and a keyboard and a pointing
15 device, e.g., a mouse or a trackball, by which the user can provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well; for example, feedback provided to the user can be any form of sensory feedback, e.g., visual
20 feedback, auditory feedback, or tactile feedback; and input from the user can be received in any form, including acoustic, speech, or tactile input.

The invention can be implemented in a computing system that includes a back-end component, e.g., as a
25 data server, or that includes a middleware component, e.g., an application server, or that includes a front-end component, e.g., a client computer having a graphical user interface or a Web browser through which a user can interact with an implementation of the
30 invention, or any combination of such back-end, middleware, or front-end components. The components of the system can be interconnected by any form or medium of digital data communication, e.g., a communication network. Examples of communication networks include a

local area network ("LAN") and a wide area network ("WAN"), e.g., the Internet.

5 The computing system can include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

10

Claims

1. A computer system (900) comprising:
 - a source file repository (100) storing a plurality
5 of active source files (AS1, AS2, AS3)
belonging to a component (C1); and
 - a central compilation service (200) that, upon
receiving (410) an activation request for at
least one inactive source file (IS1) of the
10 component (C1), compiles (420) the component
(C1) using the at least one inactive source
file (IS1) and, in case the compilation is
successfully completed, initiates (430) a
transfer (440) of the at least one inactive
15 source file (IS1) to the plurality (P1) of
active source files.
2. The computer system of claim 1, wherein the
transfer (440) each inactive source file (IS1)
20 depends on a change mode selected from the group
of:
 - adding the inactive source file (IS1) to the
plurality (P1) of active source files in case
the plurality has no corresponding active
25 source file;
 - replacing a corresponding active source file (AS1)
with the inactive source file (IS1) in case the
corresponding active source file (AS1) is
outdated; and
 - 30 deleting the corresponding active source file (AS1)
from the plurality (P1) of active source files
in case the inactive source (IS1) file is
deleted.

3. The computer system of claim 1 or 2, further comprising:
a runtime archive storage (300) to store (450) a compilation result (CR1) of the component (C1)
5 in case the compilation of the component (C1) is successfully completed.
4. The computer system of any one of the claims 1 to 3, wherein the at least one inactive source file (IS1) is stored in a further source file repository (100').
10
5. The computer system of any one of the claims 1 to 4, wherein the central compilation service (200) notifies (470) a changer (90) of the inactive source file (IS1) in case the compilation of the component fails (460).
15
6. The computer system of any one of the claims 1 to 5, wherein the central compilation service assigns a component status to the component depending on the result of the compilation, the component status being selected from the group of:
20 ready in case the compilation of the component is successfully completed; and
25 broken in case the compilation of the component fails.
7. The computer system of any one of the claims 1 to 6, wherein the central compilation service assigns a component status dirty to a further component that depends on the component in case the compilation of the component is successfully completed.
30
35

8. The computer system of any one of the claims 1 to 7, wherein the central compilation service (200) performs an incremental build when compiling (420) the component (C1) having a dependency (D2) on a further component (C2), the incremental build using a component dependency evaluator (210) to determine the dependency (D2) and to provide a previously obtained compilation result of the further component (CR2), and to provide the at least one inactive source file (IS1) and the plurality (P1) active source files of the component (C1).
9. The computer system of any one of the claims 1 to 8, wherein the central compilation service (200) performs a parallel build when compiling the component (C1) by evaluating dependencies of the component (C1) on further components and compiling the component (C1) and at least one of the further components in parallel based on the dependencies.
10. The computer system of claim 9, wherein the parallel build is performed by a cluster of build computers.

11. A central compilation service (200) comprising instructions that when loaded into a memory of a computer system (900) and executed by at least one processor of the computer system (900), perform the steps:
- 5 receiving (410) an activation request for at least one inactive source file (IS1) of the component (C1);
- 10 compiling (420) the component (C1) using the at least one inactive source file (IS1);
- in case the compilation is successfully completed, initiating (430) a transfer (440) of the at least one inactive source file (IS1) to a plurality (P1) of active source files; and
- 15 in case the compilation fails (460), notifying (470) a changer (90) of the inactive source file (IS1).
12. The central compilation service (200) of claim 11, wherein the compiling step (420) is performed as an incremental build, the component (C1) having a dependency (D2) on a further component (C2), the incremental build using a component dependency evaluator (210) to determine the dependency (D2)
- 20 and to provide a previously obtained compilation result of the further component (CR2), and to provide the at least one inactive source file (IS1) and the plurality (P1) active source files of the component (C1).
- 30

13. The central compilation service (200) of any one of the claims 11 to 12, wherein the compiling step (420) is performed as a parallel build by evaluating dependencies of the component (C1) on further components and compiling the component (C1) and at least one of the further components in parallel based on the dependencies.
14. The central compilation service (200) of claim 13, wherein the parallel build is performed by a cluster of build computers.
15. A central compilation computer (903) comprising a central compilation service (200) according to any one of the claims 11 to 14.
16. A source file activation method comprising the steps:
- receiving (410) at a central compilation service (200) an activation request for at least one inactive source file (IS1) of a component (C1);
- compiling (420) the component (C1) using the at least one inactive source file (IS1);
- in case the compilation is successfully completed, initiating (430) a transfer (440) of the at least one inactive source file (IS1) to a plurality (Pl) of active source files; and
- in case the compilation fails (460), notifying (470) a changer (90) of the inactive source file (IS1).

17. The method of claim 16, wherein the transfer (440) of each inactive source file (IS1) depends on a change mode selected from the group of:
- 5 adding the inactive source file (IS1) to the plurality (P1) of active source files in case the plurality has no corresponding active source file;
- 10 replacing an active source file (AS1) with the inactive source file (IS1) in case the corresponding active source file (AS1) is outdated; and
- 15 deleting the corresponding active source file (AS1) from the plurality (P1) of active source files in case the inactive source (IS1) file is deleted.
18. The method of claim 16 or 17, further comprising the step:
- 20 storing (450) a compilation result (CR1) of the component (C1) in a runtime archive storage (300) in case the compilation of the component (C1) is successfully completed.

19. The method of any one of the claims 16 to 18,
further comprising the steps:
assigning a component status to the component (C1)
depending on the result of the compilation, the
5 component status being selected from the group
of: ready in case the compilation of the
component is successfully completed, and broken
in case the compilation of the component fails;
and
10 assigning a component status dirty to a further
component in case the further component depends
on the component and the compilation of the
component is successfully completed.
- 15 20. The method of any one of the claims 16 to 19,
wherein the compiling step (420) is performed as an
incremental build, the component (C1) having a
dependency (D2) on a further component (C2), the
incremental build using a component dependency
20 evaluator (210) to determine the dependency (D2)
and to provide a previously obtained compilation
result of the further component (CR2), and to
provide the at least one inactive source file (IS1)
and the plurality (P1) active source files of the
25 component (C1).
21. The method of any one of the claims 16 to 20,
wherein the compiling step (420) is performed as a
parallel build by evaluating dependencies of the
30 component (C1) on further components and compiling
the component (C1) and at least one of the further
components in parallel based on the dependencies.

WO 2004/088516

PCT/EP2004/001382

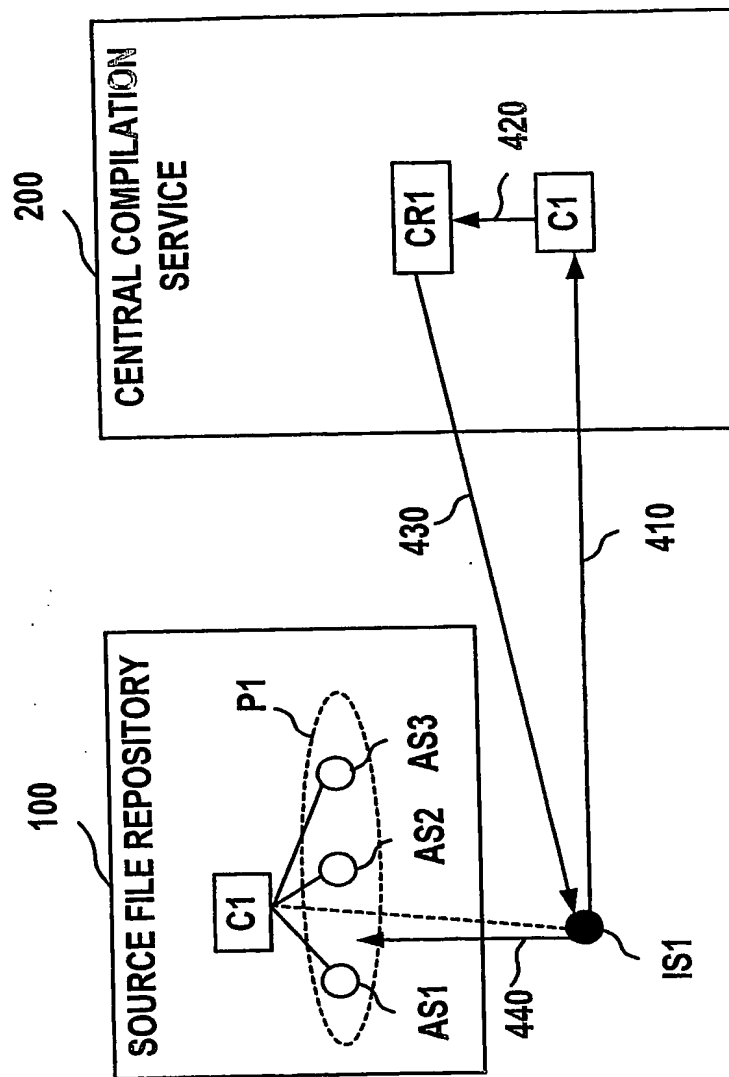
22. The method of claim 21, wherein the parallel build is performed by a cluster of build computers.

23. A method for validating software comprising the steps:
- a local file system (110) retrieving (1a) a source file of a component referencing a referenced component from a source file repository of a source control system (902);
 - the local file system (110) obtaining (1b) a compilation result of the referenced component from a runtime archive storage (RTA) (300);
 - an integrated development environment (IDE) (110) receiving a modification (EDIT) of the source file;
 - upon having received the modification, the IDE (110) transferring (2) the source file to a local build tool (140);
 - the local build tool (140) retrieving (3) the compilation result from the local file system (110);
 - the local build tool (140) locally compiling (4) the component that includes the modified source file by using the compilation result of the referenced component resulting in a new compilation result of the component;
 - storing (5) the new compilation result in the local file system (110);
 - checking in (7) the modified source file into the source file repository, the modified source file becoming an inactive source file of the source file repository;
 - launching (8) an activation request with regards to the inactive source file directed to a central compilation service (CCS) (200);

- the CCS (200) loading (9a) the inactive source file
and corresponding active source files of the
component from the source file repository, and
retrieving (9b) the compilation result of the
referenced component from the RTA 300;
the CCS 200 centrally compiling (10) the component;
and
in case of successful central compilation (10) the
CCS (200) triggering (12) the activation of the
successfully compiled inactive source file in
the source file repository.
24. The method of claim 23, comprising the further
step:
the IDE (120) deploying (6) the new compilation
result to a local runtime (130) for test
purposes prior to the checking in step (7).
25. The method of claims 23 or 24, comprising the
further step:
the CCS (200) making available (11) the result of
the central compilation in the RTA (300).
26. The method of any one of the claims 23 to 25,
comprising the further step:
the CCS (200) storing an error result in case an
error occurs during the central compilation
(10), and making available the error result to
a changer (90) of the inactive source file
causing the error.

27. A local development computer (901) configured to execute the steps retrieving (1a), obtaining (1b), transferring (2), retrieving compilation result (3), locally compiling (4), storing (5), deploying
5 (6) and launching (8) of the method according to the claims 23 or 24.

1 / 5

900**FIG. 1**

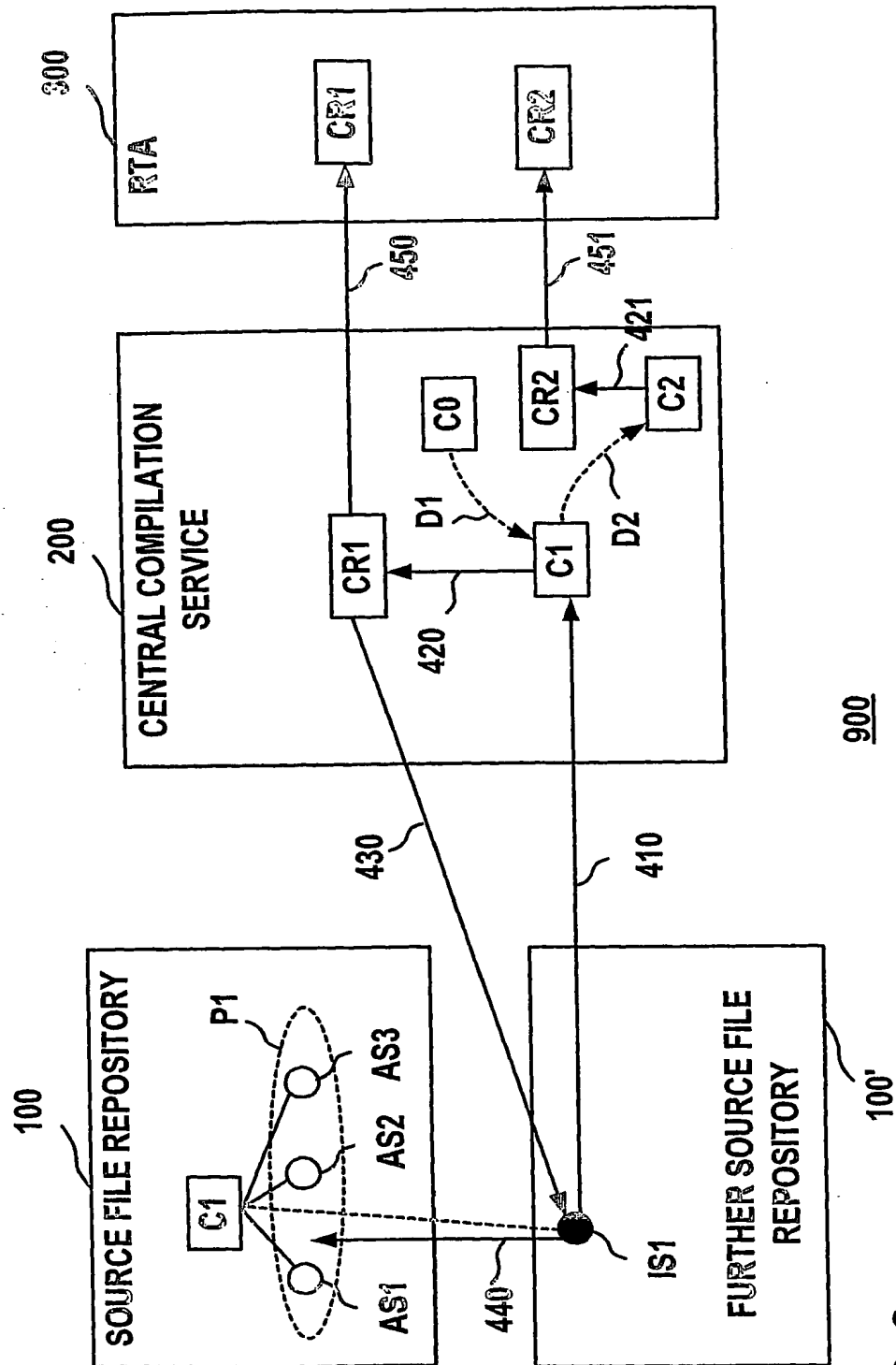


FIG. 2

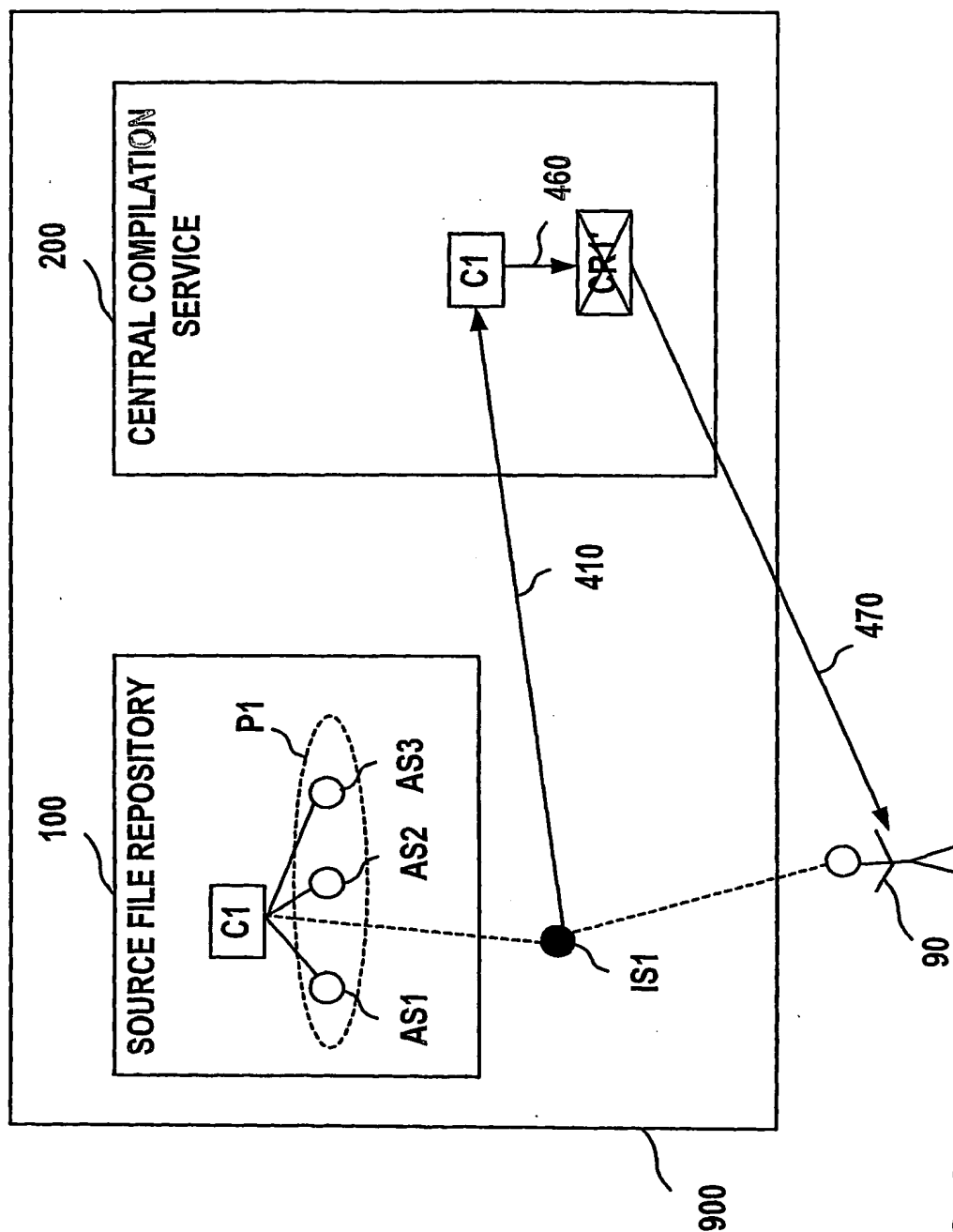


FIG. 3

4 / 5

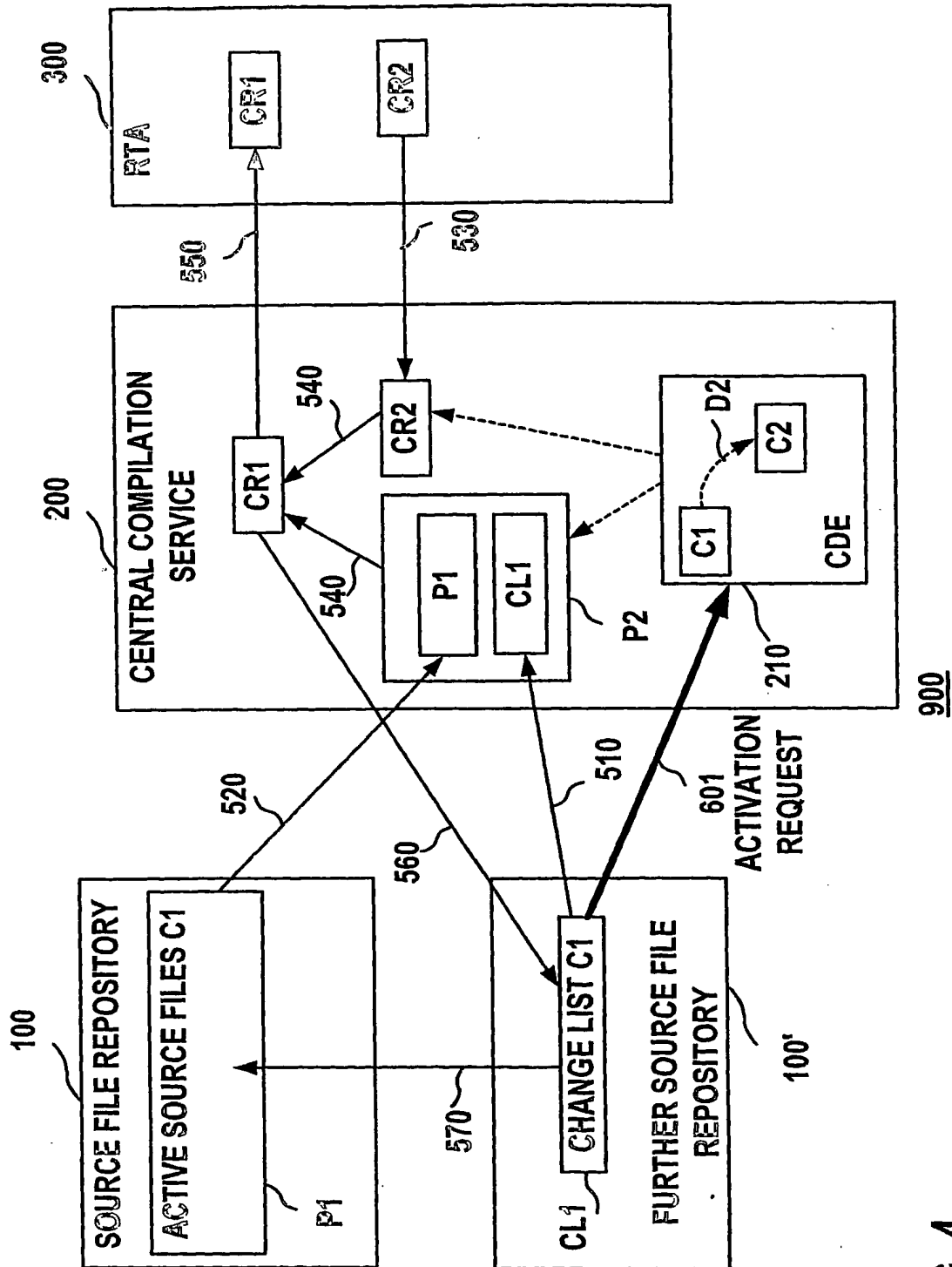


FIG. 4

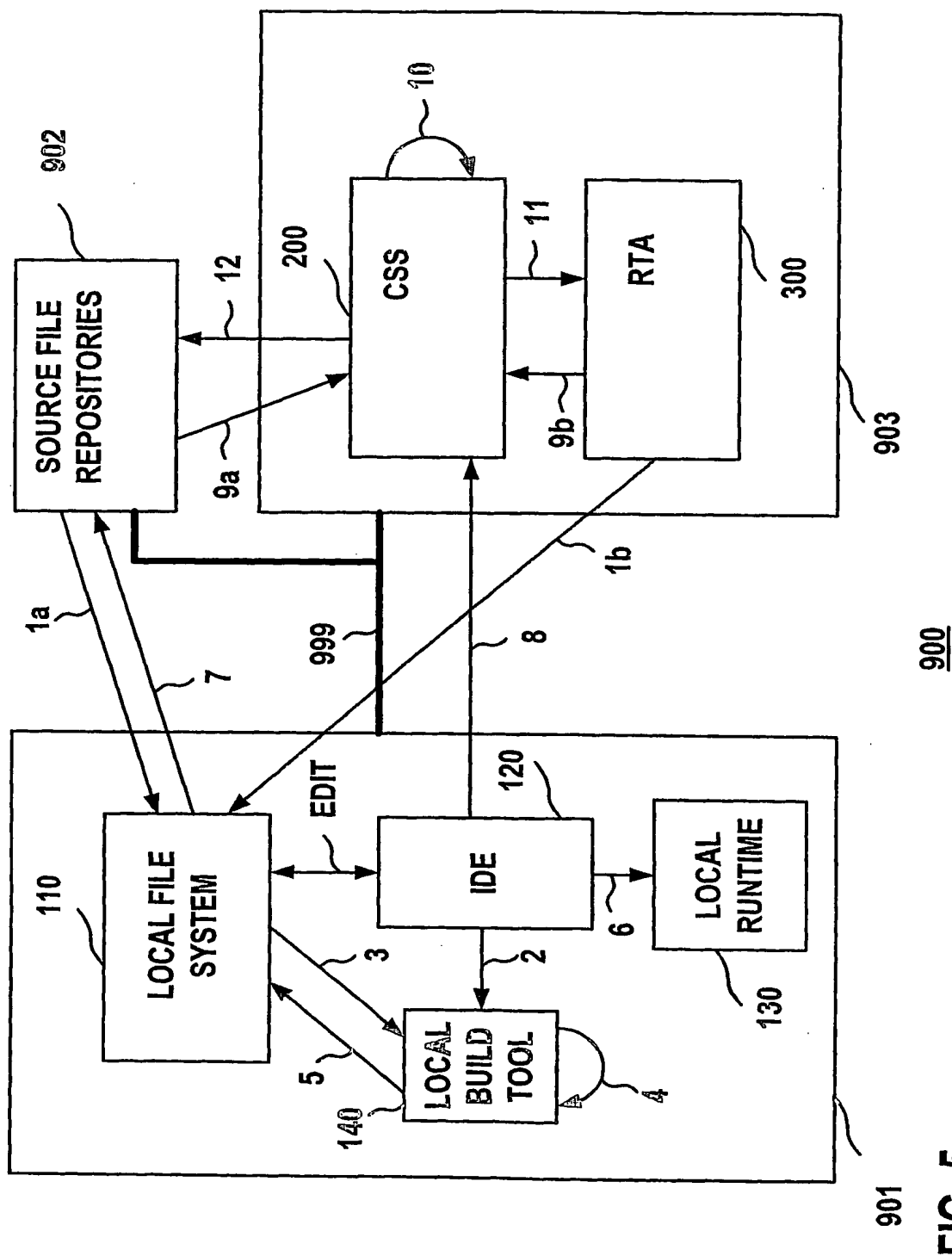


FIG. 5

INTERNATIONAL SEARCH REPORT

International Application No
EP2004/001382

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/45 G06F9/44

Rec'd PCT/PTO 26 MAY 2005

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 758 160 A (GIBBONS BILL ET AL) 26 May 1998 (1998-05-26)	1-8, 11, 12, 15-20, 23-27
Y	abstract column 2, line 13 - line 35 column 3, line 1 - line 30 column 4, line 22 - line 56 column 5, line 14 - line 45 column 5, line 65 - column 6, line 6 column 7, line 40 - line 55 column 10, line 26 - line 38 claims 1-5, 12, 21-23, 28 ----- -/--	9, 10, 13, 14, 21, 22

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- *G* document member of the same patent family

Date of the actual completion of the international search

23 June 2004

Date of mailing of the international search report

02/07/2004

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Tomàs Blanch, F

INTERNATIONAL SEARCH REPORT

International Application No
EP2004/001382

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 956 513 A (MCLAIN JR JOHN V) 21 September 1999 (1999-09-21)	1-8, 11, 12, 15-20, 23-27
Y	the whole document	9, 10, 13, 14, 21, 22
X	EP 1 202 171 A (HITACHI LTD ;HITACHI SOFTWARE ENG (JP)) 2 May 2002 (2002-05-02)	1-5, 11, 16-18, 23-27
A	abstract column 1, line 5 - line 38 column 2, line 8 - line 19 column 2, line 29 - line 52 column 3, line 56 - column 4, line 20 claims 1, 2	6-10, 12-15, 19-22
X	US 4 951 192 A (CHASE JR ROBERT P ET AL) 21 August 1990 (1990-08-21)	1, 5, 8, 11, 12, 15, 16, 20
Y	abstract	9, 10, 13, 14, 21, 22
A	claims 1-4, 10-13	2-4, 6, 7, 17-19, 23-27
Y	CHASE R P JR ET AL: "Parallel building: experience with a CASE system for workstation networks" COMPUTER WORKSTATIONS, 1988., PROCEEDINGS OF THE 2ND IEEE CONFERENCE ON SANTA CLARA, CA, USA 7-10 MARCH 1988, WASHINGTON, DC, USA, IEEE COMPUT. SOC. PR, US, 7 March 1988 (1988-03-07), pages 2-11, XP010011374 ISBN: 0-8186-0810-2 the whole document	9, 10, 13, 14, 21, 22
A	US 5 854 932 A (SPALDING DANIEL R ET AL) 29 December 1998 (1998-12-29)	8-10, 12-14, 20-22
	the whole document	
A	PER CEDERQVIST ET AL: "Version Management with CVS" SIGNUM SUPPORT AB, 'Online! 1993, XP002199439 Retrieved from the Internet: URL: http://www.loria.fr/~molli/cvs/doc/cvs .ps> 'retrieved on 2002-05-21! the whole document	1-27

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

EP2004/001382

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
US 5758160	A	26-05-1998	AU 6018794 A CA 2144874 A1 CN 1105802 A DE 69406113 D1 EP 0688448 A1 JP 8512152 T WO 9500901 A1	17-01-1995 05-01-1995 26-07-1995 13-11-1997 27-12-1995 17-12-1996 05-01-1995
US 5956513	A	21-09-1999	NONE	
EP 1202171	A	02-05-2002	JP 2002082811 A EP 1202171 A2	22-03-2002 02-05-2002
US 4951192	A	21-08-1990	CA 1302578 C	02-06-1992
US 5854932	A	29-12-1998	NONE	